Masterthesis questionnaire

Welcome!

This questionnaire is part of a master thesis written for SAP in the field of software architecture.

It will take you about 20 minutes and consists of 22 questions.

All your answers will not be correlated with you personally. The answers will only be evaluated as aggregated datasets of all participants.

If you have any questions or remarks please contact me at max.becker@sap.com.

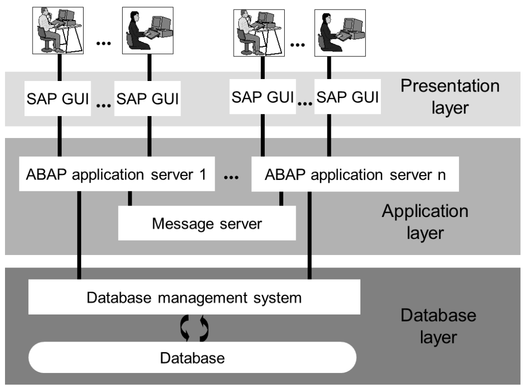
1. Optional question: **Your microservices expertise**.
In how many projects did you work with microservices? What was your role in these projects?

Defintion

This questionnaire uses the notion of *three-tier* and *microservices* architectural style.

When mentioning **a three-tier architecture** this questionnaire refers to a system like the SAP NetWeaver. It consists of three distinct layers:

- Presentation layer - interacting with the user
- Domain logic layer - handling the business logic and consisting of one or more application servers
- Data source layer - holding all the data in one central database



The presentation and the application layer can be scaled horizontally in the SAP Netweaver system. One application server can have N GUIs and multiple instances of the same application server can exist.

The **microservice architectural style** here refers to a system like the online shop amazon.com. The system consists of a suite of small services, each running in its own process and communicating with lightweight mechanisms over explicitly declared APIs. No direct database access is allowed from outside the service, and there’s no data sharing among the services. Each service has a team with up to 10 people associated with it, and that team is completely responsible for the service—from scoping out the functionality, to architecting it, to building it, and operating it. If you hit the Amazon.com gateway page more than 100 services are used to collect data and construct the site for you.

Please rate which architectural style is more suitable to achieve the given quality scenario.

2. **Resource efficiency with varying load**
Users initiate varying requests over the day to the system. The requests distribution varies around 10x of the average request count. The system elastically adapts the used hardware resources to the varying load.

microservices architecture	undecided	3-tier architecture

3. **Horizontal scaling of components**
An algorithm wishes to scale a productively running component* according to varying incoming request load. Without human interaction the system scales the component horizontally and it takes less than 5 minutes until it is scaled.

*Software component in this questionnaire refers to 'a unit of composition with contractually specified interfaces.'

microservices architecture	undecided	3-tier architecture

4. **Availability after component crash**
A component of the application crashes. Under average load the system is fault-tolerant and the functionality of the component stays available.

microservices architecture	undecided	3-tier architecture

5. **Transaction consistency**
Part of an internal system crashes during a transactional operation. The error is recorded and the system as a whole goes back to a consistent state. During no time the system is available and in an inconsistent state.

microservices architecture	undecided	3-tier architecture

6. **Data consistency**
Two users initiate a request with the same input at the same time under normal operations both users consistently receive the same answer.

microservices architecture	undecided	3-tier architecture

7. **Modifiability on component level**
A development team wishes to change a component with the best of breed in the given area. Technology stack and programming language should not be limiting factors for the choice. It will take less than one week to test and deploy the change into production.

microservices architecture	undecided	3-tier architecture

8. **Reuse of software component**
Another project wants to reuse a logical component which is implemented in the system. The component can be reused in different contexts/programs.

microservices architecture	undecided	3-tier architecture

9. **Data segregation**
An attacker tries to access sensitive data within the system after gaining control of a relatively uncritical part of the system. The system denies access to more sensitive data and logs the attempt.

microservices architecture	undecided	3-tier architecture

10. **Installability - on premise**

A customer wants to decide whether he deploys the system on premise. The system can be deployed on the local infrastructure of the customer.

microservices architecture	undecided	3-tier architecture

Organization Philosophy

Please rate which architectural style works better together with the given *organizational* requirement.

11. **Developer Velocity and Continious Delivery**
The organization demands that the cycle-time is less than a day. Meaning that a code change (no interface changes) affecting a single component is automatically deployed, tested and run into production in less than a day.

microservices architecture	undecided	3-tier architecture

12. **Q-Gates**
The organization requires to review every software change through a thorough (>1week) quality process before it goes into production.

microservices architecture	undecided	3-tier architecture

13. **Technical Teams**
The organization demands to organize teams around technical qualification leading to pure UI, deployment or operation teams.

microservices architecture	undecided	3-tier architecture

14. **Explicit interfaces**
All communication between different components (internal or external) is only allowed over explicitly defined APIs. No other way of communication between components is possible (for example over a database).

microservices architecture	undecided	3-tier architecture

SAP development philosophy

Please rate the SAP development philosophy. You need to specify a tendency.

15. **Developer Velocity and Continuous Delivery**
The organization demands that the cycle-time is less than a day. Meaning that a code change (no interface changes) affecting a single component is automatically deployed, tested and run into production in less than a day.

not the SAP way — all the time at SAP

16. **Q-Gates**
The organization requires to review every software change through a thorough (>1week) quality process before it goes into production.

not the SAP way — all the time at SAP

17. **Technical Teams**
The organization demands to organize teams around technical qualification leading to pure UI, deployment or operation teams.

not the SAP way — all the time at SAP

18. **Explicit interfaces**
All communication between different components (internal or external) is only allowed over explicitly defined APIs. No other way of communication between components is possible (for example over a database).

not the SAP way — all the time at SAP

Application Scenario: ERP system with HR management

Imagine you would design the software architecture for an ERP system with HR management on a green field.

19. Please order the following quality attributes by importance starting with the most important one to the least important one for an **ERP system with HR management**. 1 = most important, 9 = least important.

These are the same quality scenarios you rated earlier. An explanatory scenario for each quality attribute is given below.

	Resource efficiency with varying load
	Horizontal scaling of components
	Availability after component crash
	Transaction consistency
	Data Consistency
	Modifiability on component level
	Reuse of a software component
	Data segregation
	Installability - on premise

<u>Example scenarios</u>

Resource efficiency with varying load
Users initiate varying requests over the day to the system. The requests distribution varies around 10x of the average request count. The system elastically adapts the used hardware resources and scale

Horizontal scaling of components
An algorithm wishes to scale a productively running component according to varying incoming request load. Without human interaction the system scales the component elastically and it takes less than 5 minutes until it is scaled.

Availability after component crash
A component of the application crashes. Under average load the system is fault-tolerant and the functionality of the component stays available.

Transaction consistency
Part of an internal system crashes during a transactional operation. The error is recorded and the system as a whole goes back to a consistent state. During no time the system is available and in an inconsistent state.

Data consistency
Two users initiate a request with the same input at the same time under normal operations both users consistently receive the same answer.

Modifiability on component level
A development team wishes to change a component with the best of breed in the given area. Technology stack and programming language should not be limiting factors for the choice. It will take less than one week to test and deploy the change into production.

Reuse of software component
Another project wants to reuse a logical component which is implemented in the system. The component can be reused in different contexts/programs.

Data segregation
An attacker tries to access sensitive data within the system after gaining control of a relatively uncritical part of the system. The system denies access to more sensitive data and logs the attempt.

Installability - on premise
A customer wants to decide whether he deploys the system on premise. The system can be deployed on the local infrastructure of the customer.

Masterthesis questionnaire

Application Scenario: Social Network

Imagine you would design the software architecture for a social network like Facebook on a green field.

20. Please order the following quality attributes by importance starting with the most important one to the least important one for a **social network**, like Facebook. 1 = most important, 9 = least important.

An explanatory scenario for each quality attribute is given below.

	Resource efficiency with varying load
	Horizontal scaling of components
	Availability after component crash
	Transaction consistency
	Data Consistency
	Modifiability on component level
	Reuse of a software component
	Data segregation
	Installability - on premise

Explanation

Resource efficiency with varying load
Users initiate varying requests over the day to the system. The requests distribution varies around 10x of the average request count. The system elastically adapts the used hardware resources and scale

Horizontal scaling of components
An algorithm wishes to scale a productively running component according to varying incoming request load. Without human interaction the system scales the component elastically and it takes less than 5 minutes until it is scaled.

Availability after component crash
A component of the application crashes. Under average load the system is fault-tolerant and the functionality of the component stays available.

Transaction consistency
Part of an internal system crashes during a transactional operation. The error is recorded and the system as a whole goes back to a consistent state. During no time the system is available and in an inconsistent state.

Data consistency
Two users initiate a request with the same input at the same time under normal operations both users consistently receive the same answer.

Modifiability on component level
A development team wishes to change a component with the best of breed in the given area. Technology stack and programming language should not be limiting factors for the choice. It will take less than one week to test and deploy the change into production.

Reuse of software component
Another project wants to reuse a logical component which is implemented in the system. The component can be reused in different contexts/programs.

Data segregation
An attacker tries to access sensitive data within the system after gaining control of a relatively uncritical part of the system. The system denies access to more sensitive data and logs the attempt.

Installability - on premise
A customer wants to decide whether he deploys the system on premise. The system can be deployed on the local infrastructure of the customer.

Masterthesis questionnaire

Application Scenario: E-Commerce

Imagine you would design the software architecture for an e-commerce system, like amazon.com on a green field.

21. Please order the following quality attributes by importance starting with the most important one to the least important one for an **e-commerce system**, like amazon. 1 = most important, 9 = least important.

An explanatory scenario for each quality attribute is given below.

	Resource efficiency with varying load
	Horizontal scaling of components
	Availability after component crash
	Transaction consistency
	Data Consistency
	Modifiability on component level
	Reuse of a software component
	Data segregation
	Installability - on premise

Explanation

Resource efficiency with varying load
Users initiate varying requests over the day to the system. The requests distribution varies around 10x of the average request count. The system elastically adapts the used hardware resources and scale

Horizontal scaling of components
An algorithm wishes to scale a productively running component according to varying incoming request load. Without human interaction the system scales the component elastically and it takes less than 5 minutes until it is scaled.

Availability after component crash
A component of the application crashes. Under average load the system is fault-tolerant and the functionality of the component stays available.

Transaction consistency
Part of an internal system crashes during a transactional operation. The error is recorded and the system as a whole goes back to a consistent state. During no time the system is available and in an inconsistent state.

Data consistency
Two users initiate a request with the same input at the same time under normal operations both users consistently receive the same answer.

Modifiability on component level
A development team wishes to change a component with the best of breed in the given area. Technology stack and programming language should not be limiting factors for the choice. It will take less than one week to test and deploy the change into production.

Reuse of software component
Another project wants to reuse a logical component which is implemented in the system. The component can be reused in different contexts/programs.

Data segregation
An attacker tries to access sensitive data within the system after gaining control of a relatively uncritical part of the system. The system denies access to more sensitive data and logs the attempt.

Installability - on premise
A customer wants to decide whether he deploys the system on premise. The system can be deployed on the local infrastructure of the customer.

Masterthesis questionnaire

End of survey

If you have additional comments or want to get a copy of the thesis once it is done please contact me at max.becker@sap.com.

Thank you for participating!